



Creación de Aplicaciones Python

Lección 11: Frameworks GUI [2/2]: PyQt

ÍNDICE

Lección 11. – Frameworks GUI [2/2]- PyQt	2
Presentación y objetivos	2
1. PyQt5: Conceptos Básicos	3
2. PyQt6: Versión más reciente	8
3. PyQt con uso de la Función Main y POO	12
4. Qt Designer: Primeros pasos	14
5. Uso de código QSS en Qt Designer	17
6. Importar el Diseño (.ui) desde Atom con Python	24
7. QSS programado en el propio PyQt	30
8. Puntos clave	35

Lección 11. – Frameworks GUI

[2/2]- PyQt

PRESENTACIÓN Y OBJETIVOS


En la pasada lección vimos el Framework GUI Tkinter, en esta ocasión estaremos viendo el Framework PyQt.

En esta ocasión nos iremos a la versión 6, PyQt6 y haremos algún ejemplo con PyQt5.

PyQt a fecha de Abril 2021, es uno de los Frameworks GUI más populares. Para trabajar con este Framework disponemos de una herramienta llamada “QtDesigner” que nos ahorrará mucho tiempo en el diseño inicial.

INSTALACIÓN

Ya fue vista en la primera lección



Objetivos

- Conocer los fundamentos de PyQt
- Conocer Qt Designer
- Ver algunas diferencias PyQt5 y PyQt6

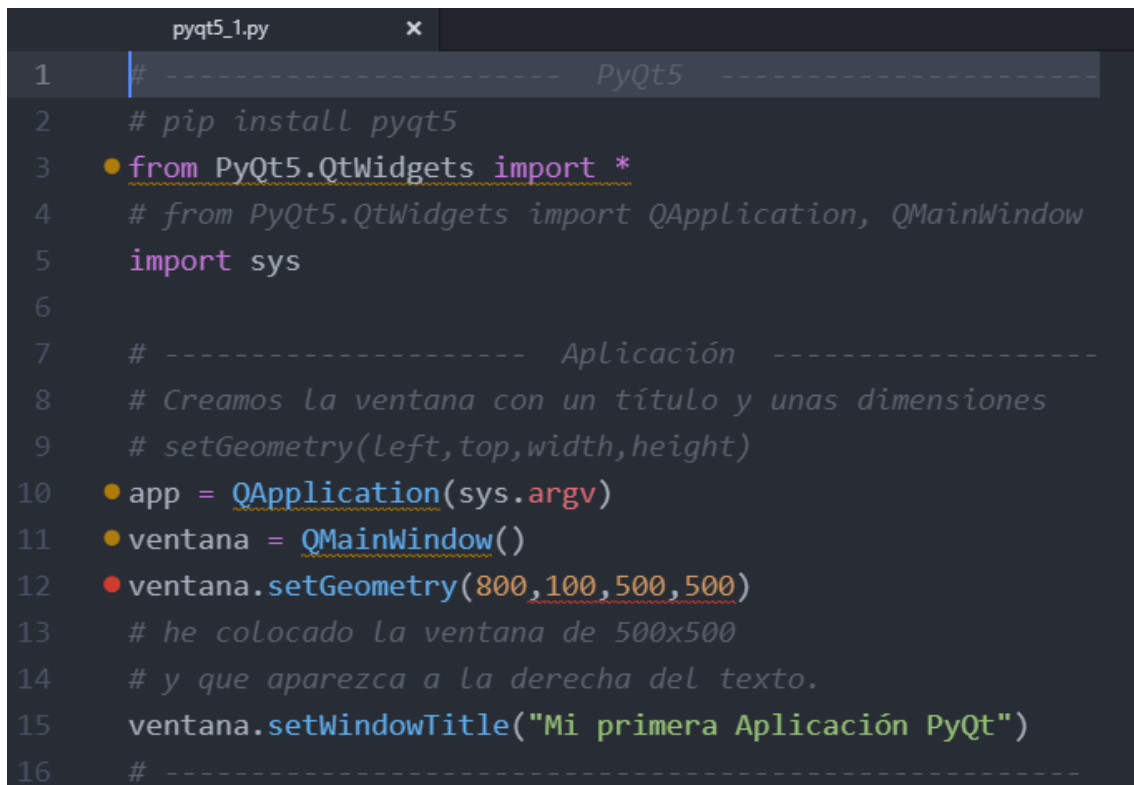
Comencemos !

1. PYQT5: CONCEPTOS BÁSICOS

Comenzaremos explicando una pequeña aplicación programada con PyQt5, aunque pronto pasaremos a PyQt6.

Si importamos `*` significa que importamos todo.

En la línea siguiente hemos dejado con comentario que podría haberse importado nuestras dependencias de esa forma.



```
pyqt5_1.py x
1  # ----- PyQt5 -----
2  # pip install pyqt5
3  • from PyQt5.QtWidgets import *
4  # from PyQt5.QtWidgets import QApplication, QMainWindow
5  import sys
6
7  # ----- Aplicación -----
8  # Creamos la ventana con un título y unas dimensiones
9  # setGeometry(left,top,width,height)
10 • app = QApplication(sys.argv)
11 • ventana = QMainWindow()
12 • ventana.setGeometry(800,100,500,500)
13 # he colocado la ventana de 500x500
14 # y que aparezca a la derecha del texto.
15 ventana.setWindowTitle("Mi primera Aplicación PyQt")
16 # -----
```

Figura 1.1: PyQt5_parte 1

Aclaraciones de la anterior figura:

- | Importamos todo lo relativo a PyQt5.QtWidgets
- | Importamos sys lo usaremos más adelante
- | Indicamos una ventana de 500x500 que aparecerá a la derecha (800,100)
- | Le indicamos un título

```

17  # ----- labels -----
18  # Creamos una etiqueta con un texto y en una ubicación
19  ● label_1 = QLabel(ventana)
20    label_1.setText("Aplicación programada con PyQt5 !")
21    label_1.adjustSize()
22    label_1.move(150, 20)
23
24  ● label_2 = QLabel(ventana)
25    label_2.setText("Escribe algo en el siguiente QLineEdit: ")
26    label_2.adjustSize()
27    label_2.move(20, 100)
28  # -----

```

Figura 1.2: PyQt5_parte 2

Aclaraciones de la anterior figura:

- | Creamos 2 Labels para la ventana.
- | Indicamos un texto para cada una
- | Le decimos que se ajuste (en tamaño) al texto
- | Le indicamos la posición, siendo (0,0) arriba a la izquierda

```

29  # ----- funcion_imprimir -----
30  # función asociada a un botón (boton_1)
31  # combinamos con line_edit_1
32  ● def funcion_imprimir():
33      print(line_edit_1.text())
34
35  # ----- funcion salir -----
36  # función asociada al botón_2
37  # Le decimos que salga de la ventana.
38  ● def funcion_salir():
39      ventana.close()
40  # -----

```

Figura 1.3: PyQt5_parte 3

Aclaraciones de la anterior figura:

- | Defino 2 funciones que me harán falta posteriormente
- | Una de ellas es para imprimir en la "cmd"
- | La otra función sirve para salir.

```
41 # ----- botones -----
42 # Botón para imprimir
43 • boton_1 = QPushButton(ventana)
44   boton_1.setText("Imprimir en cmd")
45   boton_1.clicked.connect(funcion_imprimir)
46   boton_1.move(20, 200)
47
48 # botón para salir de la aplicación
49 • boton_2 = QPushButton(ventana)
50   boton_2.setText("Salir")
51   boton_2.clicked.connect(funcion_salir)
52   boton_2.move(390, 450)
53
```

Figura 1.4: PyQt5_parte 4

Aclaraciones de la anterior figura:

- | Creo 2 botones (y les ubicaré en la ventana)
- | El primero de ellos lo conecto con la función_imprimir
- | El segundo de ellos sirve para salir de la aplicación

```
54 # ----- line_edit -----
55 • line_edit_1 = QLineEdit(ventana)
56   line_edit_1.move(20, 150)
57
```

Figura 1.5: PyQt5_parte 5

Aclaraciones de la anterior figura:

- | Sirve para hacer una entrada de texto.
- | La misma será impresa en "cmd" gracias a su vinculación con el botón de imprimir y la propia función asociada a ese botón

```
58 # ----- show y exit -----
59 ventana.show()
60 sys.exit(app.exec_())
```

Figura 1.6: PyQt5_parte 6

Aclaraciones de la anterior figura:

- | Muestra la ventana y cierra la ventana
- | También podemos cerrar la Aplicación con el aspa (x) de cerrar

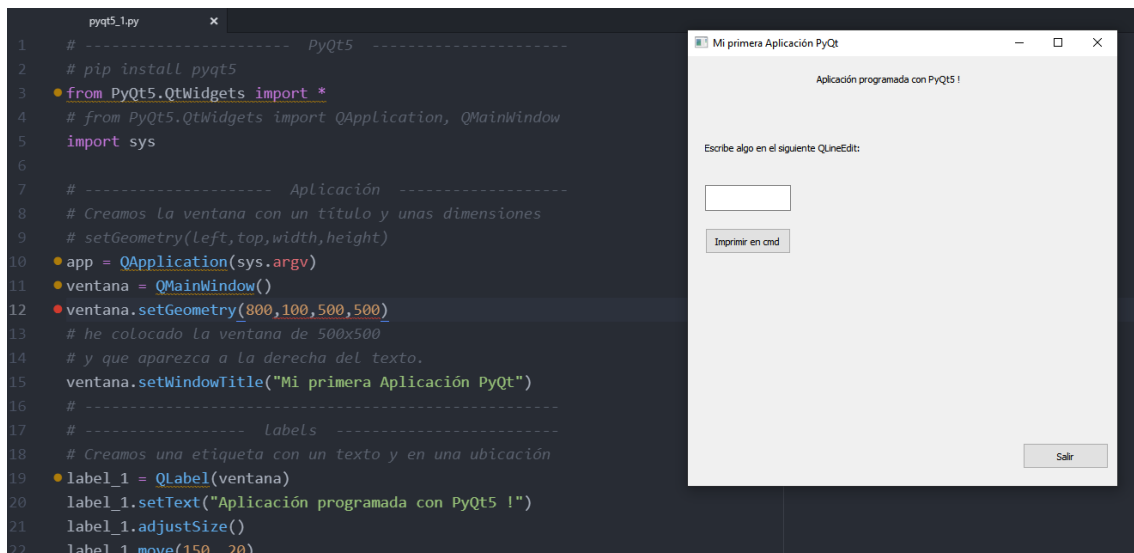


Figura 1.7: PyQt5_parte 7

Aclaraciones de la anterior figura:

- | Tal y como hemos indicado la aplicación sale justo a la derecha
- | La ubicación de cada elemento en la ventana es la esperada.

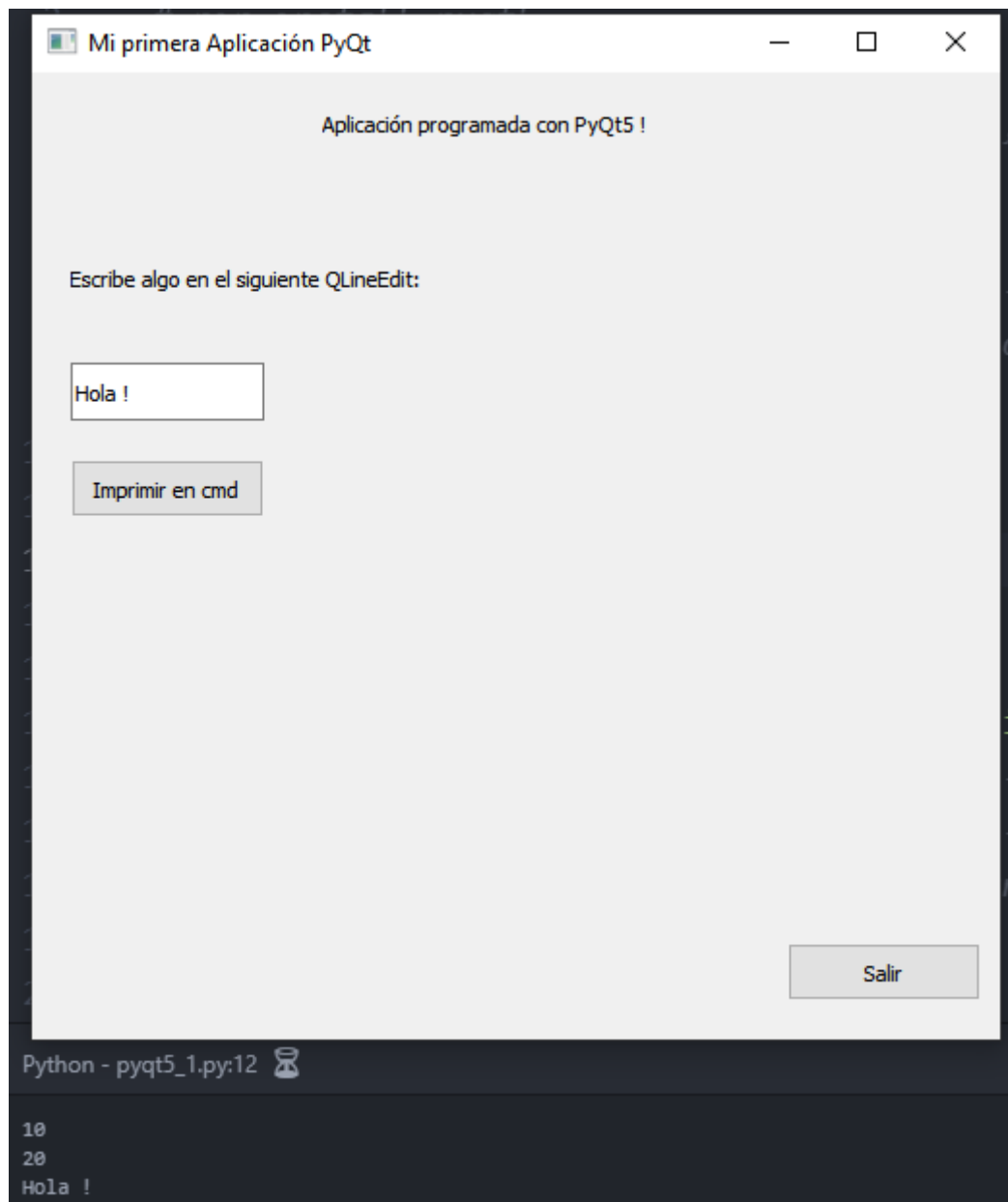


Figura 1.8: PyQt5_parte 8

Aclaraciones de la anterior figura:

- | Hemos escrito "10" y pulsado en "Imprimir en cmd"
- | Después hemos hecho lo mismo para "20" y para "Hola !"
- | Vemos que imprime números enteros, decimales e incluso strings.

2. PYQT6: VERSIÓN MÁS RECIENTE

Lo ideal al trabajar en Software es trabajar con la versión más reciente y estable.

En este caso, trataremos de realizar el mismo código que hicimos con PyQt5 pero con PyQt6.

A la hora de hacer este tipo de cambios es probable que haya modificaciones en las propias librerías.

Si nos vamos a la propia web:

<https://www.riverbankcomputing.com/static/Docs/PyQt6/>

PyQt v6.0 Reference Guide » PyQt6 Reference Guide



Next topic

[Introduction](#)

Quick search

Go

PyQt6 Reference Guide

- Introduction
 - License
 - PyQt6 Components
- Contributing to this Documentation
 - Repository Structure
 - Description Files
 - Contributing Patches
- Support for Old Versions of Python
- Installing PyQt6
 - Understanding the Correct Version to Install
 - Installing from Wheels
 - Building and Installing from Source
- Differences Between PyQt6 and PyQt5
- Support for Signals and Slots
 - Unbound and Bound Signals
 - Defining New Signals with `pyqtSignal`
 - Connecting, Disconnecting and Emitting Signals
 - Connecting Signals Using Keyword Arguments
 - The `pyqtSlot` Decorator
 - The `PyQt_PyObject` Signal Argument Type
 - Connecting Slots By Name
- Support for Qt Properties
 - Defining New Qt Properties
- Other Support for Dynamic Meta-objects
 - `pyqtEnum`
 - `pyqtClassInfo`
- Support for Qt Interfaces
- Support for QVariant

Figura 2.1: PyQt6_Reference Guide

Y hacemos click en: “Differences Between PyQt6 and PyQt5” veremos lo siguiente:

Differences Between PyQt6 and PyQt5

In this section we give an overview of the differences between PyQt6 and PyQt5. This is not an exhaustive list and does not go into the detail of the differences between the Qt v6 and Qt v5 APIs.

- All named enums are now implemented as a sub-class of the standard Python `Enum` class. (PyQt5 used `IntEnum` for scoped enums and a custom type for traditional named enums).
- Qt provides the `QFlags` template class as a type-safe way of using enum values that can be combined as a set of flags. The name of the class is often the plural form of the name of the enum. PyQt5 implements both of these as separate types. PyQt6 instead combines them as a single type, using the plural name, as a sub-class of `Flag`.
- `Q_CLASSINFO()` has been replaced by the `pyqtClassInfo()` class decorator.
- `Q_ENUM()`, `Q_ENUMS()`, `Q_FLAG()` and `Q_FLAGS()` have been replaced by the `pyqtEnum()` class decorator.
- All `exec_()` and `print_()` methods have been removed.
- `QApp` has been removed.
- The `PyQT_CONFIGURATION` dict has been removed.
- The `qt` module has been removed.
- The bindings for the (GPL licensed) Qt classes that implement support for network authorisation have moved out to a separate add-on project `PyQt6-NetworkAuth`. This means that all of the libraries wrapped by PyQt6 itself are licensed under the LGPL.
- `pylupdate6` is a completely new pure-Python implementation. It can no longer read a `.pro` file in order to determine the names of `.py` files to translate.
- Support for Qt's resource system has been removed (i.e. there is no `pyrcc6`).
- Python v3.6.1 or later is required.

Qt v6 implements a number of functions from Qt v5 that are now marked as being deprecated. These are not supported in PyQt6.

Figura 2.2: PyQt6_diferencias con PyQt5

De modo que el método `exec_()` ha sido eliminado.

En este caso se ha encontrado una solución simple al mismo. (simplemente quitando la barra baja). Tal y como veremos a continuación e donde la última línea será: `sys.exit(app.exec())`

A su vez ahora importaremos PyQt6: “pip install PyQt6” en la “cmd”

```
C:\Users\Manut>pip install PyQt6
Collecting PyQt6
  Downloading PyQt6-6.0.3-cp36-cp37-cp38-cp39-none-win_amd64.whl (5.2 MB)
    |#####| 5.2 MB 6.4 MB/s
Collecting PyQt6-Qt6>=6.0
  Downloading PyQt6_Qt6-6.0.2-py3-none-win_amd64.whl (41.8 MB)
    |#####| 41.8 MB 62 kB/s
Collecting PyQt6-sip<14,>=13.0
  Downloading PyQt6_sip-13.0.1-cp38-cp38-win_amd64.whl (57 kB)
    |#####| 57 kB 126 kB/s
Installing collected packages: PyQt6-sip, PyQt6-Qt6, PyQt6
Successfully installed PyQt6-6.0.3 PyQt6-Qt6-6.0.2 PyQt6-sip-13.0.1

C:\Users\Manut>
```

Figura 2.3: PyQt6_Ejemplo de instalación desde la “cmd”

```

pyqt_v6_2.py
1  # ----- PyQt6 -----
2  # pip install PyQt6
3  ● from PyQt6.QtWidgets import *
4  import sys
5  # ----- Aplicación -----
6  # Creamos la ventana con un título y unas dimensiones
7  # setGeometry(left,top,width,height)
8  ● app = QApplication(sys.argv)
9  ● ventana = QMainWindow()
10 ● ventana.setGeometry(800,100,500,500)
11 # he colocado la ventana de 500x500
12 # y que aparezca a la derecha del texto.
13 ventana.setWindowTitle("Mi primera Aplicación PyQt")
14 # ----- Labels -----
15 # Creamos una etiqueta con un texto y en una ubicación
16 ● label_1 = QLabel(ventana)
17 label_1.setText("Aplicación programada con PyQt6 !")
18 label_1.adjustSize()
19 label_1.move(150, 20)
20

```

Figura 2.4: PyQt6_Aplicación (parte 1)

Aclaraciones de la anterior figura:

- | Hemos cambiado PyQt5 por PyQt6 en todos los casos.
- | El resto lo hemos dejado como estaba.

```

21  • label_2 = QLabel(ventana)
22  label_2.setText("Escribe algo en el siguiente QLineEdit: ")
23  label_2.adjustSize()
24  label_2.move(20, 100)
25  # ----- funcion_imprimir -----
26  # función asociada a un botón (boton_1)
27  # combinamos con line_edit_1
28  • def funcion_imprimir():
29  |     print(line_edit_1.text())
30  # ----- funcion salir -----
31  # función asociada al botón_2
32  # le decimos que salga de la ventana.
33  • def funcion_salir():
34  |     ventana.close()
35  # ----- botones -----
36  # Botón para imprimir
37  • boton_1 = QPushButton(ventana)
38  boton_1.setText("Imprimir en cmd")
39  boton_1.clicked.connect(funcion_imprimir)
40  boton_1.move(20, 200)

```

Figura 2.5: PyQt6_Aplicación (parte 2)

Aclaraciones de la anterior figura:

- No hemos hecho cambios en esta parte de código

```

41  # botón para salir de la aplicación
42  • boton_2 = QPushButton(ventana)
43  boton_2.setText("Salir")
44  boton_2.clicked.connect(funcion_salir)
45  boton_2.move(390, 450)
46  # ----- line_edit -----
47  • line_edit_1 = QLineEdit(ventana)
48  line_edit_1.move(20, 150)
49  # ----- show y exit -----
50  ventana.show()
51  sys.exit(app.exec())

```

Figura 2.6: PyQt6_Aplicación (parte 3)

Aclaraciones de la anterior figura:

- Solo hemos hecho cambios en la última línea tal y como habíamos comentado.

3. PYQT CON USO DE LA FUNCIÓN MAIN Y POO

Para explicarlo, usaremos el mismo ejemplo, pero en este caso emplearemos la Función Main.

```

1  # ----- PyQt6 -----
2  • from PyQt6.QtWidgets import *
3  import sys
4  # ----- funcion_imprimir -----
5  • def funcion_imprimir():
6      print(line_edit_1.text())
7  # ----- funcion_salir -----
8  • def funcion_salir():
9      ventana.close()

```

Figura 3.1: PyQt6 con Función Main (parte 1)

```

10 # ----- Función Main -----
11 • if name == "main":
12     app = QApplication(sys.argv)
13     ventana = QMainWindow()
14     • ventana.setGeometry(800,100,500,500)
15     ventana.setWindowTitle("Usando la Función main")
16     # ----- Labels -----
17     # Label_1
18     • label_1 = QLabel(ventana)
19     label_1.setText("Aplicación programada con PyQt6 !")
20     label_1.adjustSize()
21     label_1.move(150, 20)
22     # Label_2
23     • label_2 = QLabel(ventana)
24     label_2.setText("Escribe algo en el siguiente QLineEdit: ")
25     label_2.adjustSize()
26     label_2.move(20, 100)

```

Figura 3.2: PyQt6 con Función Main (parte 2)

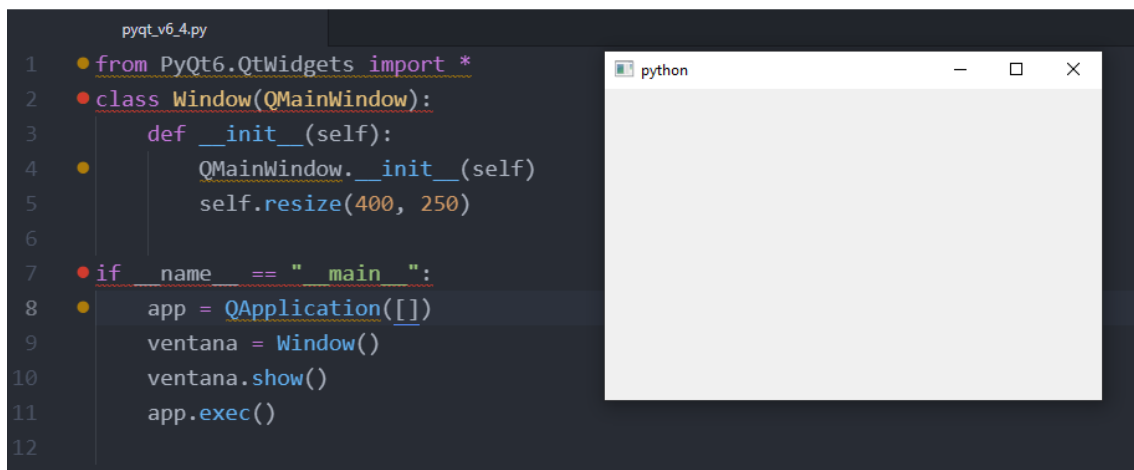
```

27 # ----- botones -----
28 # Botón para imprimir
29 • boton_1 = QPushButton(ventana)
30 boton_1.setText("Imprimir en cmd")
31 boton_1.clicked.connect(funcion_imprimir)
32 boton_1.move(20, 200)
33 # botón para salir de la aplicación
34 • boton_2 = QPushButton(ventana)
35 boton_2.setText("Salir")
36 boton_2.clicked.connect(funcion_salir)
37 boton_2.move(390, 450)
38 # ----- line_edit -----
39 • line_edit_1 = QLineEdit(ventana)
40 line_edit_1.move(20, 150)
41 # ----- show y exit -----
42 ventana.show()
43 sys.exit(app.exec())

```

Figura 3.3: PyQt6 con Función Main (parte 3)

O incluso podríamos orientarlo a objetos.



The screenshot shows a code editor with a file named `pyqt_v6_4.py`. The code defines a `Window` class that inherits from `QMainWindow`. The `__init__` method calls `QMainWindow.__init__(self)` and `self.resize(400, 250)`. The `if __name__ == '__main__':` block creates a `QApplication` instance, instantiates the `Window` class, calls `show()`, and then `app.exec()`. To the right of the code editor, a window titled "python" is visible, showing a blank white area, which represents the running application.

```

pyqt_v6_4.py
1 • from PyQt6.QtWidgets import *
2 • class Window(QMainWindow):
3     def __init__(self):
4         QMainWindow.__init__(self)
5         self.resize(400, 250)
6
7 • if __name__ == '__main__':
8     app = QApplication([])
9     ventana = Window()
10    ventana.show()
11    app.exec()
12

```

Figura 3.4: PyQt6 con Función Main (parte 4)

4. QT DESIGNER: PRIMEROS PASOS

Hasta entonces hemos ejecutando la aplicación en el propio IDE.

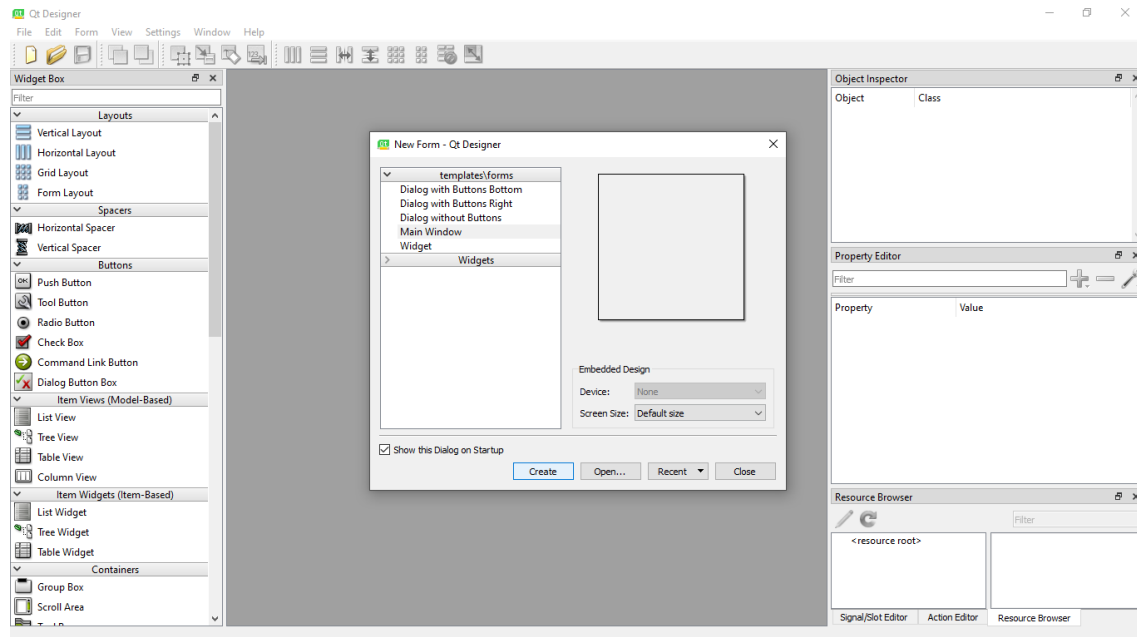


Figura 4.1: Qt Designer (parte 1)

Seleccionamos “Main Window”

Podemos, si queremos quitar el texto que nos aparece arriba. Es una barra de tareas.

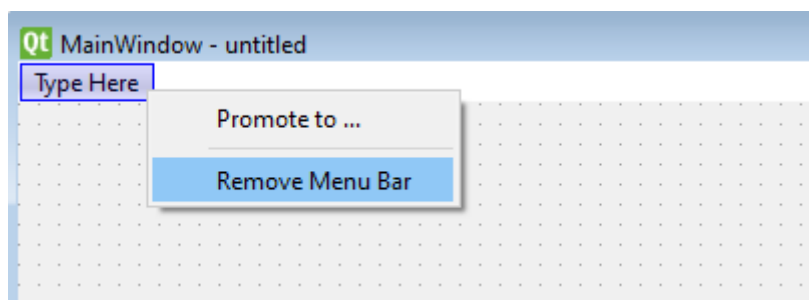


Figura 4.2: Qt Designer (parte 2)

Podemos seleccionar un botón, el cual colocaremos donde queramos, y cambiarlo el tamaño, el texto, etc.

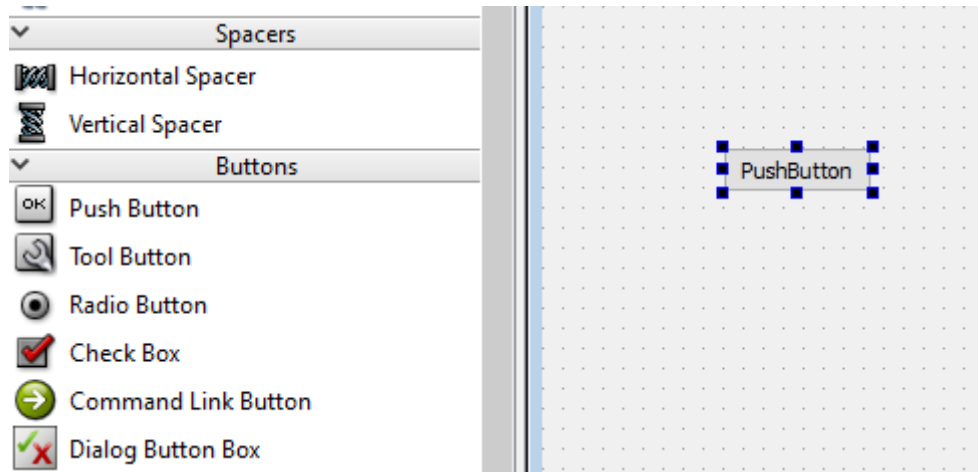


Figura 4.3: Qt Designer (parte 3)

Por ejemplo, le podemos cambiar el Texto y el "objectName"

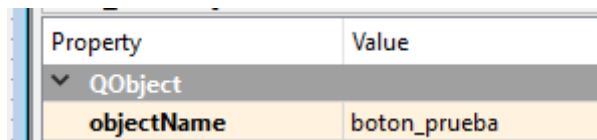


Figura 4.4: Qt Designer (parte 4)

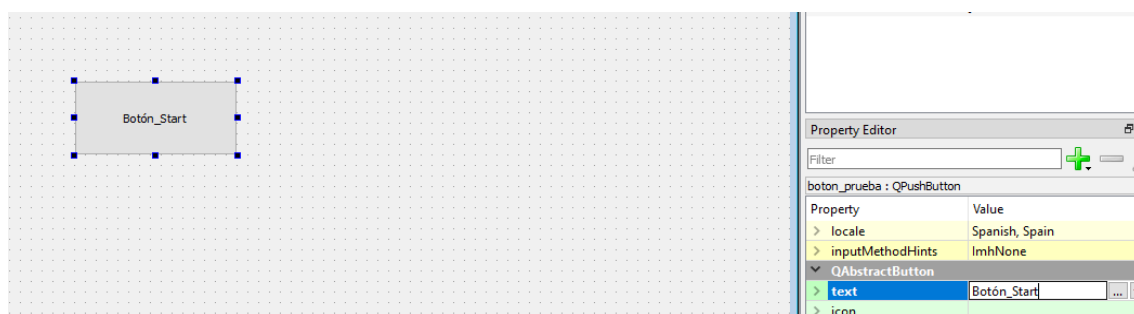


Figura 4.5: Qt Designer (parte 5)

Pudiéramos seleccionar una etiqueta (una label) y hacer diferentes cosas, de la misma manera.

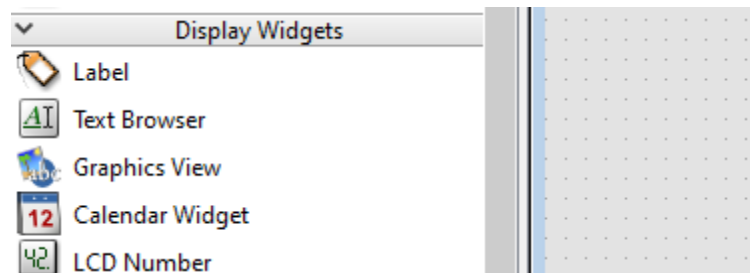


Figura 4.6: Qt Designer (parte 6)

Si queremos guardarlo:

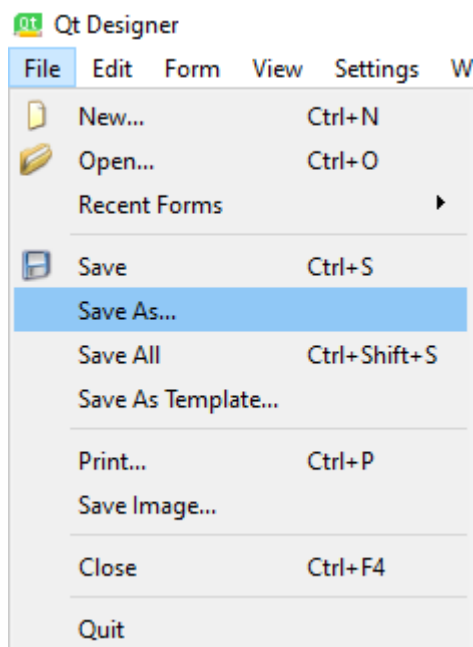


Figura 4.7: Qt Designer (parte 7)

Elegimos la carpeta que queramos, le indicamos un nombre, y la extensión por defecto, en este caso es “.ui”.

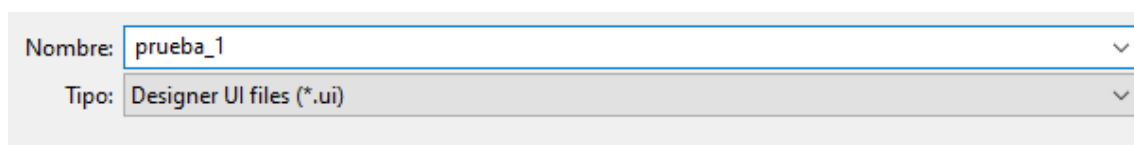


Figura 4.8: Qt Designer (parte 8)

Posteriormente veremos qué hacer con este archivo .ui

5. USO DE CÓDIGO QSS EN QT DESIGNER

Es posible añadir a las Aplicaciones código QSS (Qt Style Sheets), y está basado como se ha comentado en CSS.

Para el caso de Qt Designer se hace de la siguiente manera.

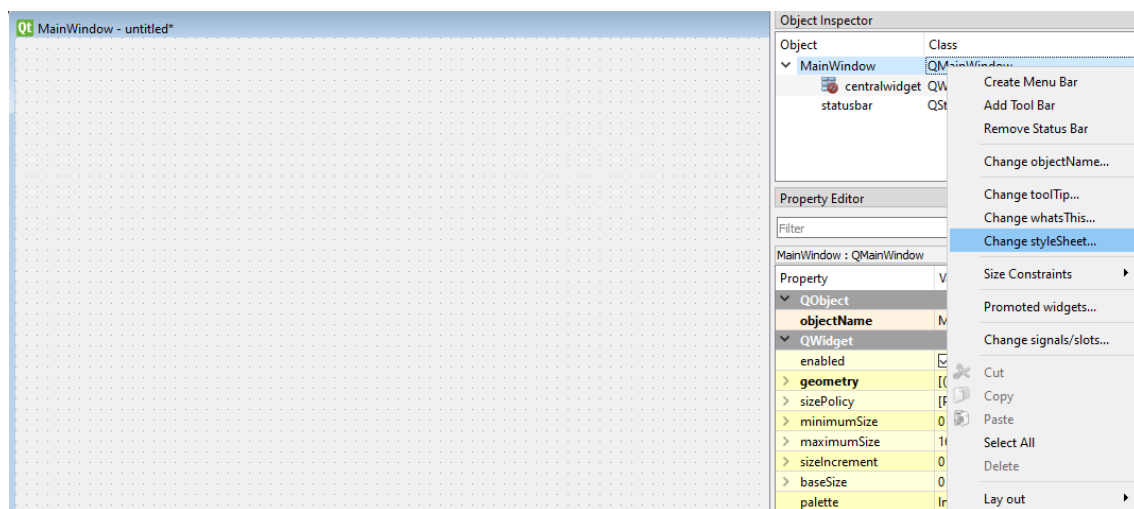


Figura 5.1: QSS en Qt Designer (Parte 1)

Y una vez ahí:

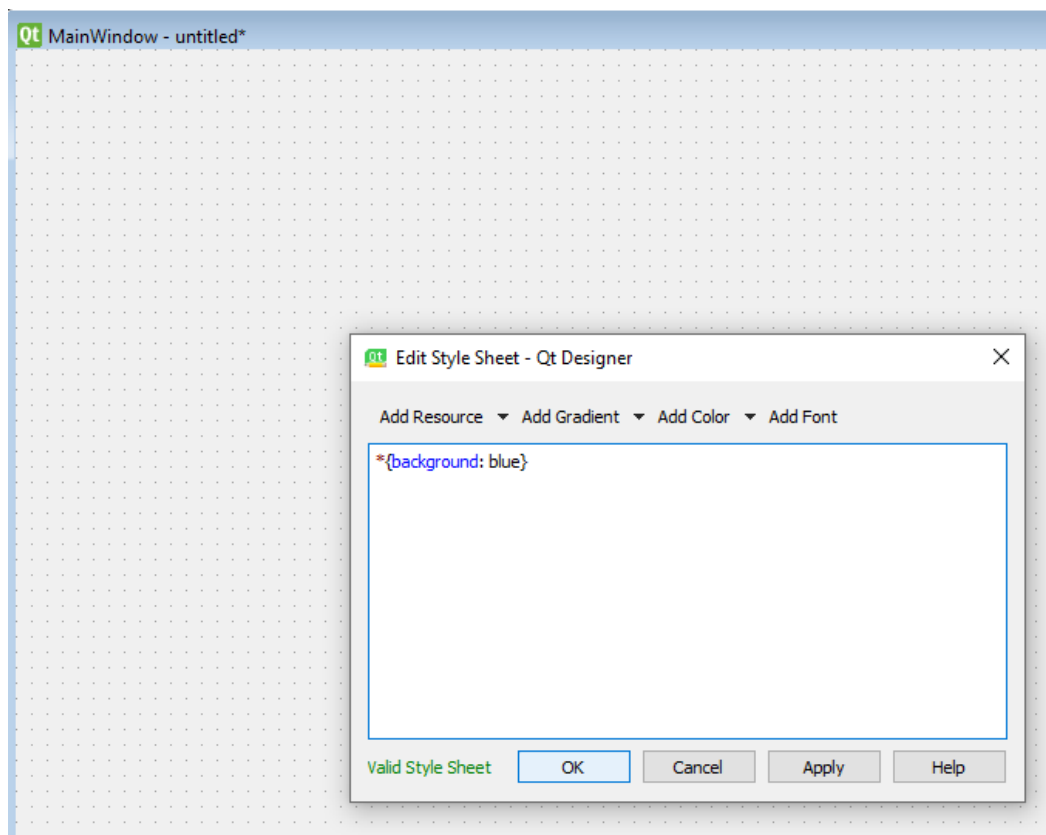


Figura 5.2: QSS en Qt Designer (Parte 2)

Al pulsar OK:

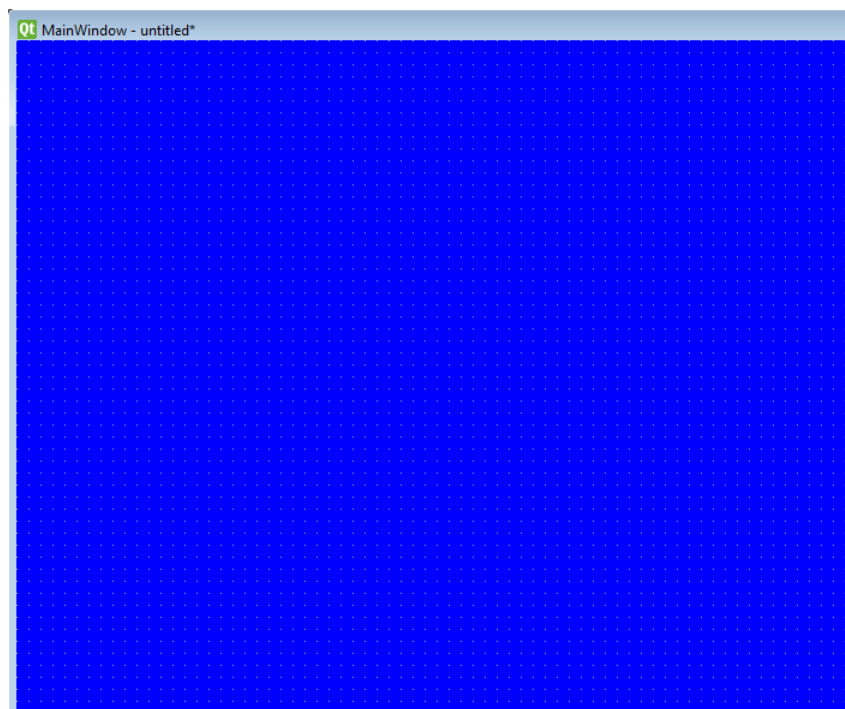


Figura 5.3: QSS en Qt Designer (Parte 3)

Podemos hacer más cambios.

También podríamos añadir un label, y emplear el código QSS

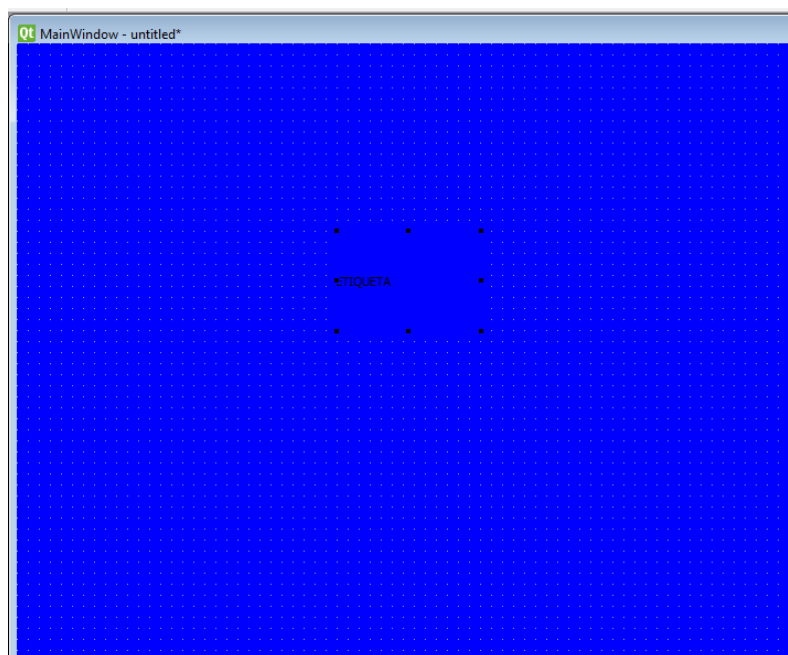


Figura 5.4: QSS en Qt Designer (Parte 4)

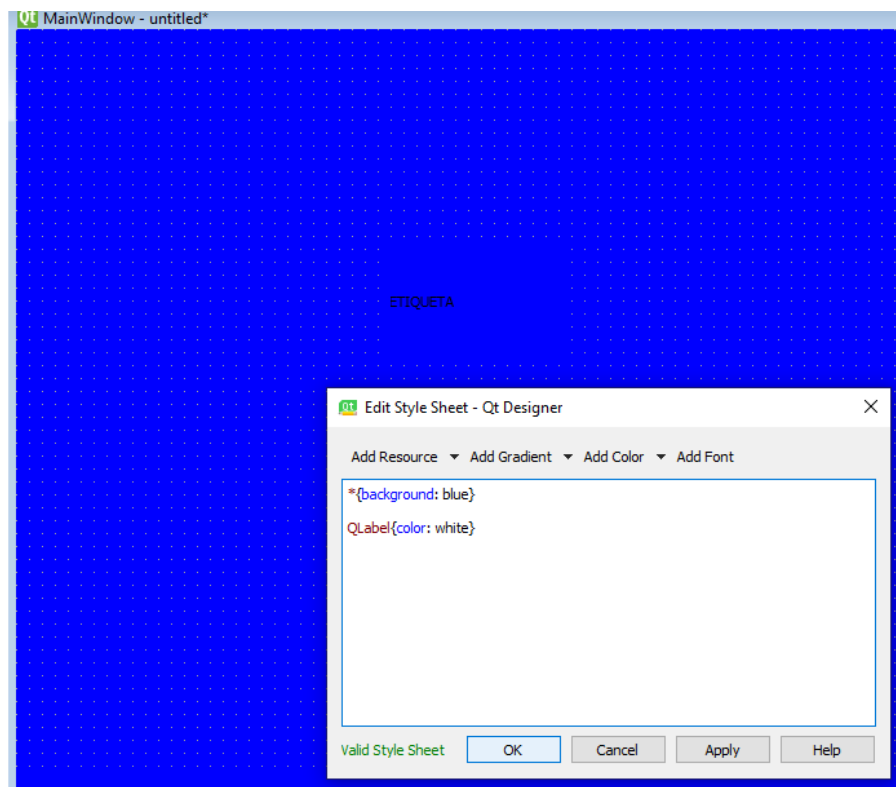


Figura 5.5: QSS en Qt Designer (Parte 5)

Y al pulsar OK:

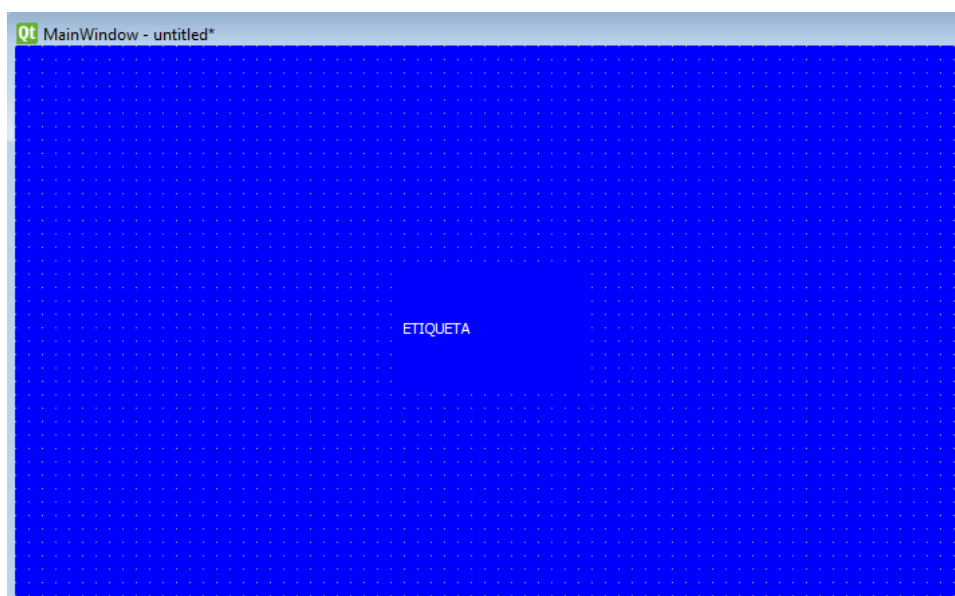


Figura 5.6: QSS en Qt Designer (Parte 6)

También se puede:

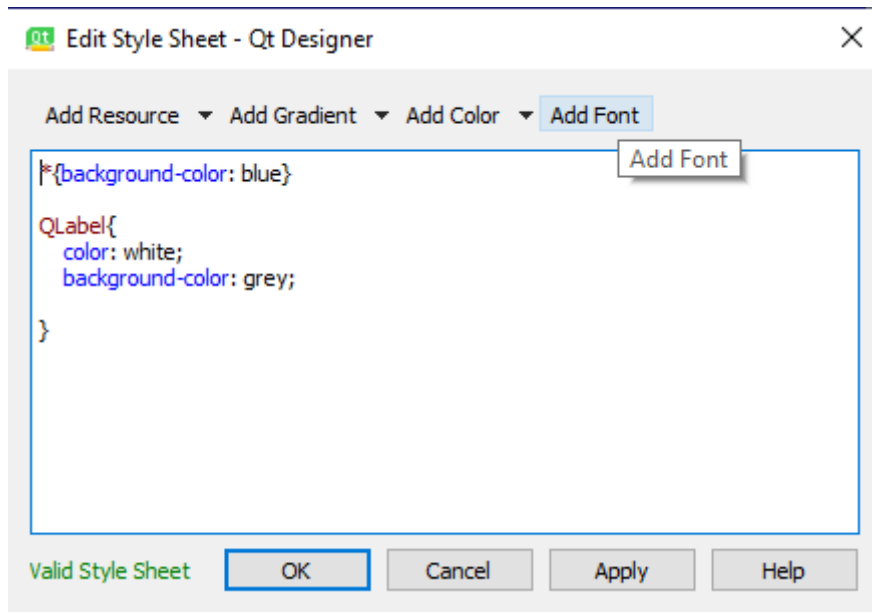


Figura 5.7: QSS en Qt Designer (Parte 7)

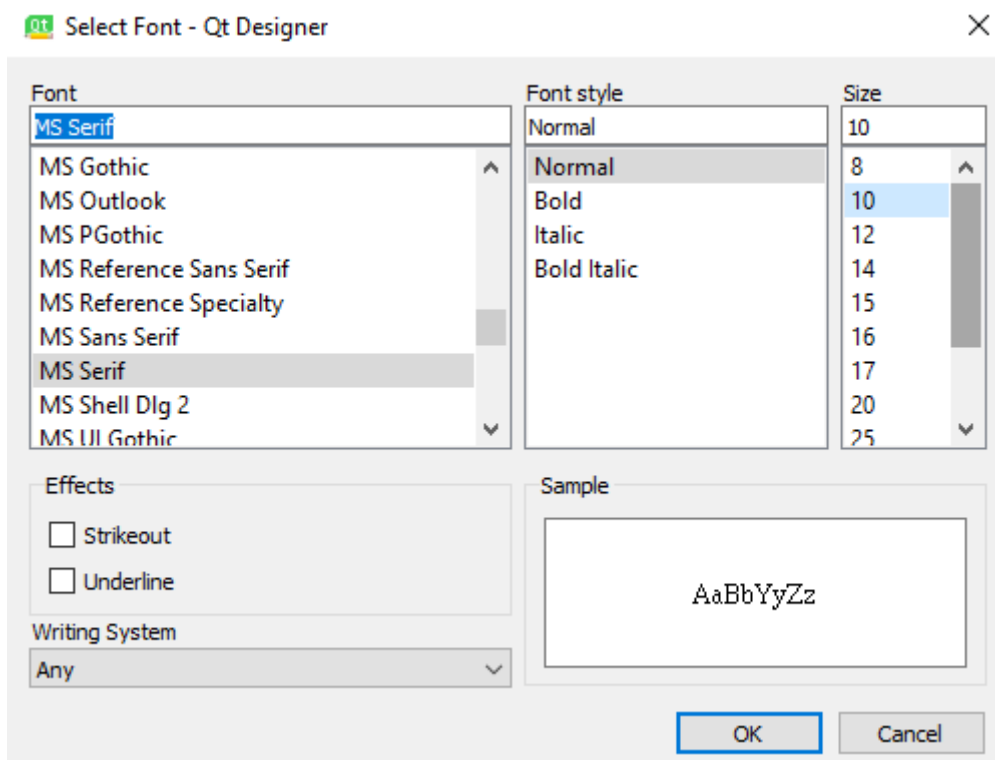


Figura 5.8: QSS en Qt Designer (Parte 8)

Y cambiaría el código, le decimos : “Aplicar” y en ese momento veríamos los cambios.

Y una vez estemos conformes le indicamos : “OK”.

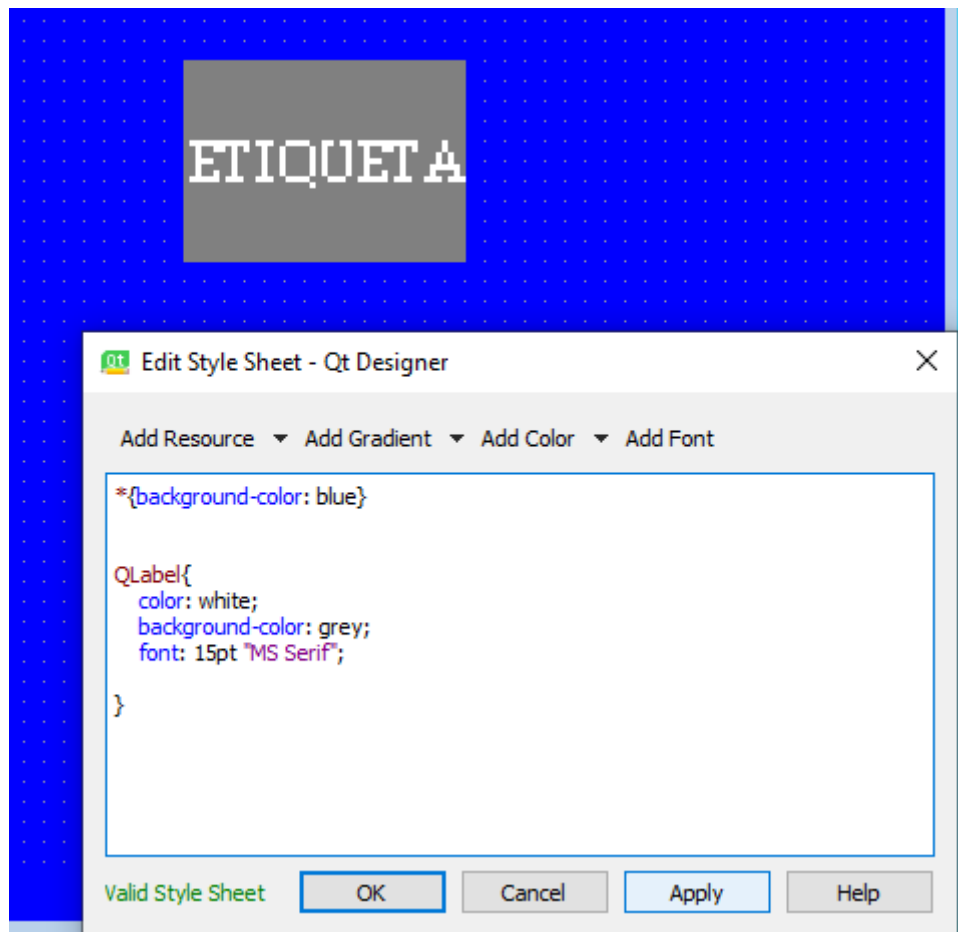


Figura 5.9: QSS en Qt Designer (Parte 9)

Si queremos centrar el texto, a mí no me funciona el: text-align: center

Por lo que he buscado alternativa y es esta:

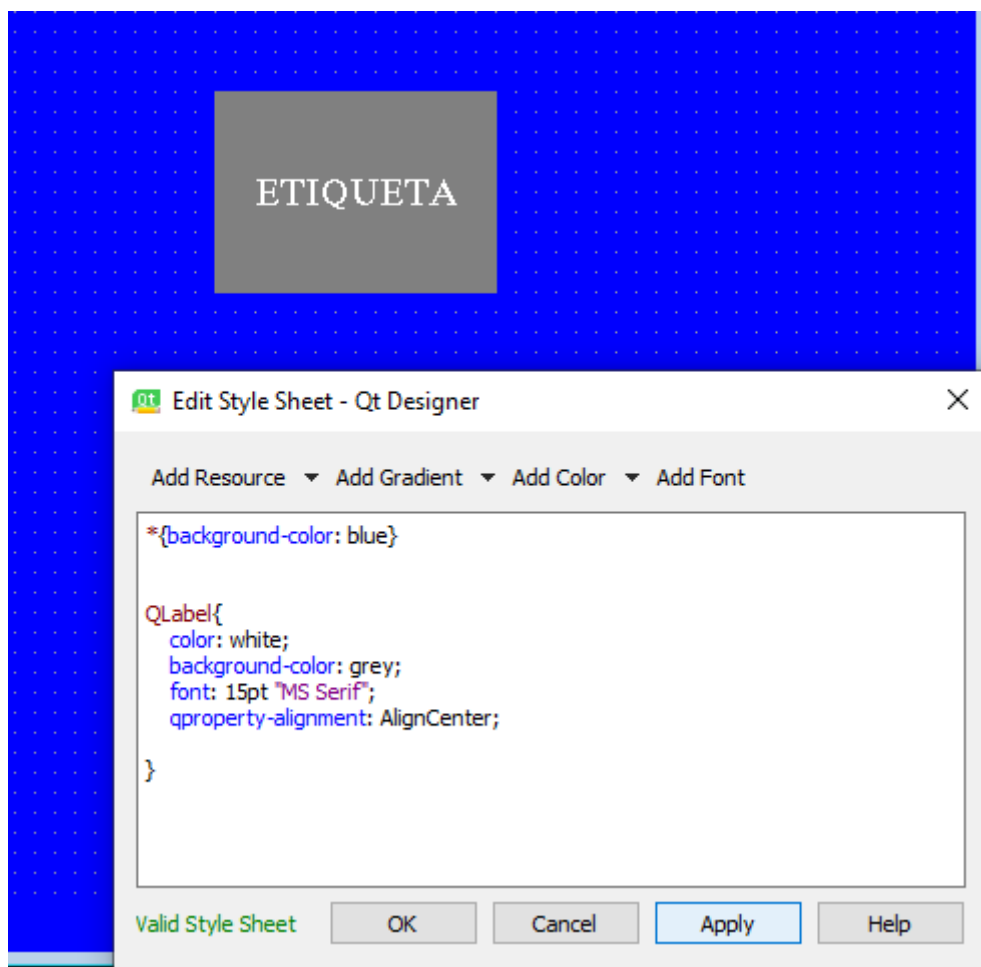


Figura 5.10: QSS en Qt Designer (Parte 10)

6. IMPORTAR EL DISEÑO (.UI) DESDE ATOM CON PYTHON

Hemos hecho un diseño con Qt Designer.

No es el mejor de los diseños, pero ahora tiene otra apariencia.

Con Qt Designer y con el uso de QSS podemos hacer diseños más personalizables.



Figura 6.1: Importar el .ui con Python (Parte 1)

Para ello hemos hecho uso de los elementos del menú que se encuentra a la izquierda:

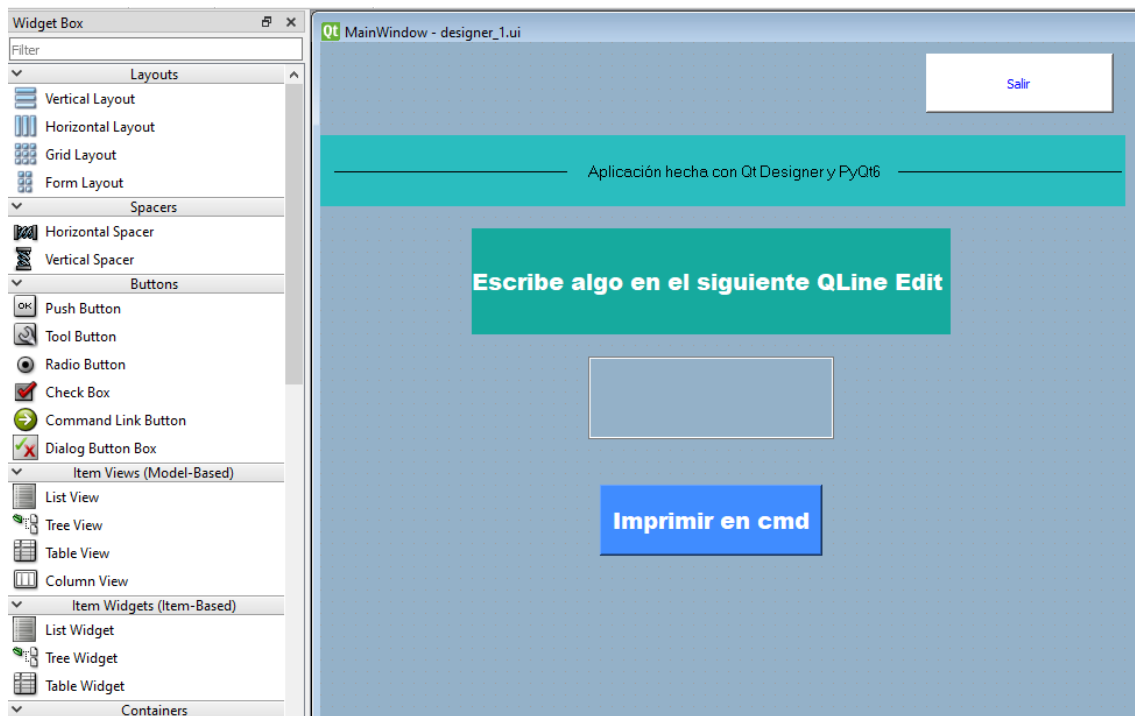


Figura 6.2: Importar el .ui con Python (Parte 2)

A continuación se detallarán los QSS:

QSS Principal

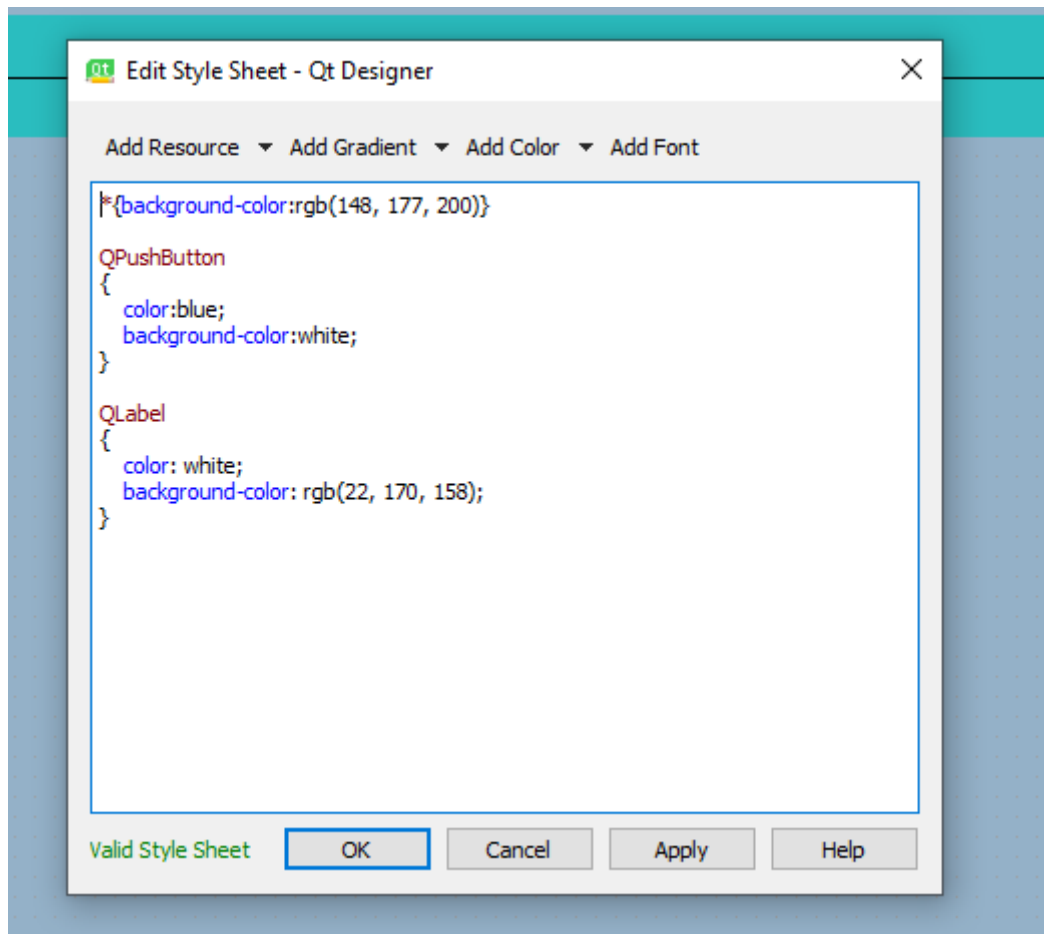


Figura 6.3: Importar el .ui con Python (Parte 3)

QSS del botón “imprimir en cmd”

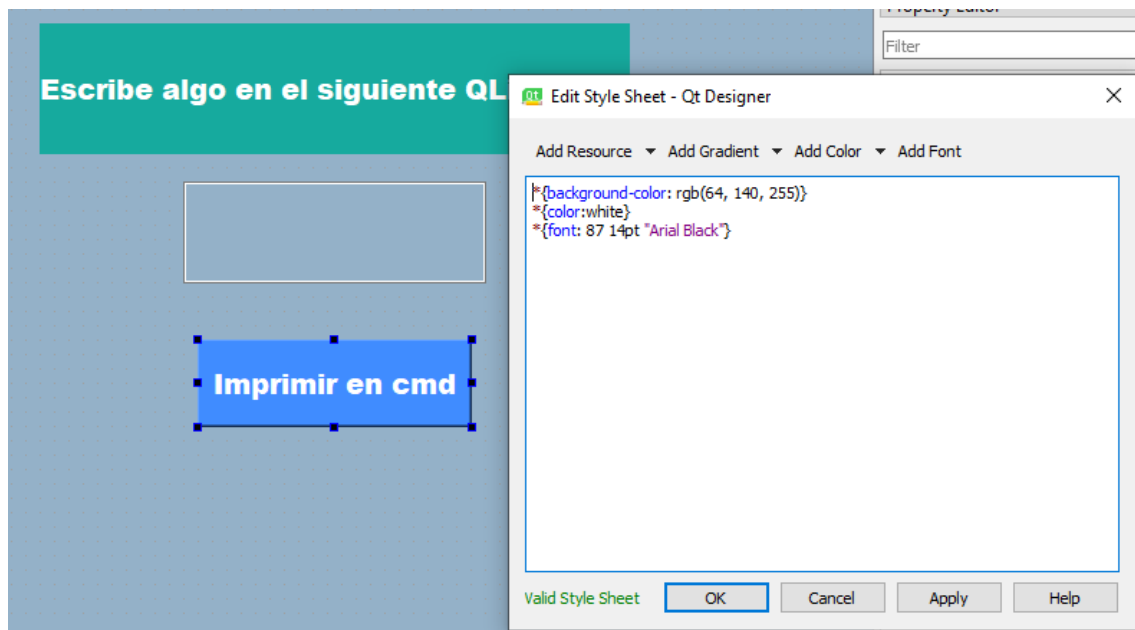


Figura 6.4: Importar el .ui con Python (Parte 4)

QSS del Label: "Escribe algo en el siguiente.."

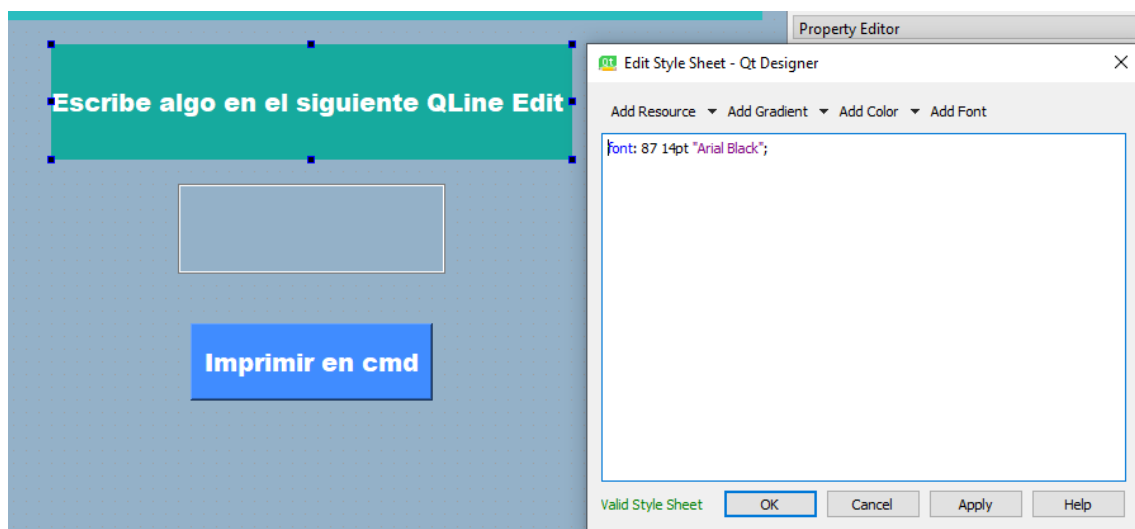


Figura 6.5: Importar el .ui con Python (Parte 5)

QSS del Label: "Aplicación hecha con..."

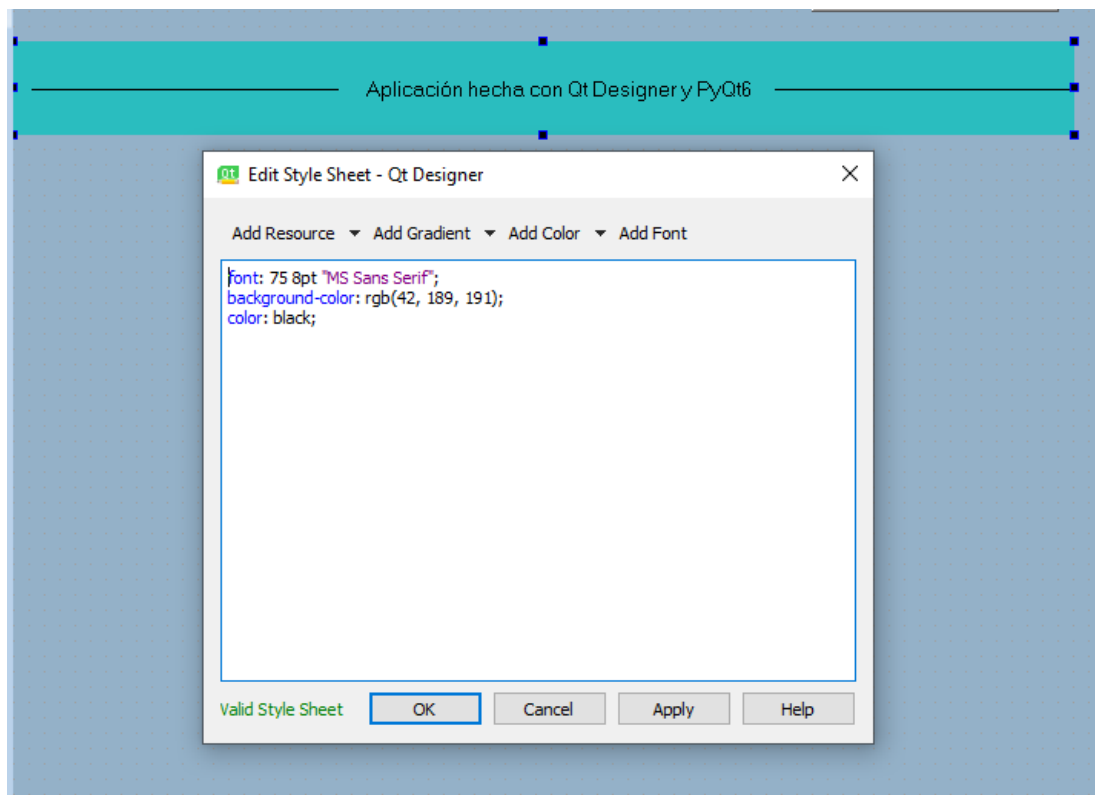


Figura 6.6: Importar el .ui con Python (Parte 6)

El resto de elementos no tiene un QSS asociado.

Ahora nos vamos a ATOM,

Y escribimos un código con Programación Orientada a Objetos (POO), como se indica a continuación, el objetivo inicial será reconocer el diseño, para lo cual importaremos el archivo .ui.

```

1  import sys
2  from PyQt6 import uic
3  from PyQt6.QtWidgets import *
4
5  class ejemplo_GUI(QMainWindow):
6      def __init__(self):
7          super().__init__()
8          # copiamos el path donde tenemos el archivo .ui (designer_1.ui)
9          # C:\Users\Manut\Desktop\apuntes_web\TEMA_11\proyectos\proyectos_QtDesigner
10         uic.loadUi("C:/Users/Manut/Desktop/apuntes_web/TEMA_11/proyectos/proyectos_QtDesigner/designer_1.ui", self)
11
12     if name == " main ":
13         app = QApplication(sys.argv)
14         GUI = ejemplo_GUI()
15         GUI.show()
16         sys.exit(app.exec())

```

Figura 6.7: Importar el .ui con Python (Parte 7)

Que si ejecutamos obtendremos:



Figura 6.8: Importar el .ui con Python (Parte 8)

Por el momento nuestros elementos no tienen funcionalidad.

Lo siguiente que haremos será indicarle esa funcionalidad.

De forma similar a lo explicado en anteriores temas.

7. QSS PROGRAMADO EN EL PROPIO PYQT

Tal y como hemos visto, podemos hacer una aplicación usando Programación Orientada a Objetos (POO) de la siguiente manera:

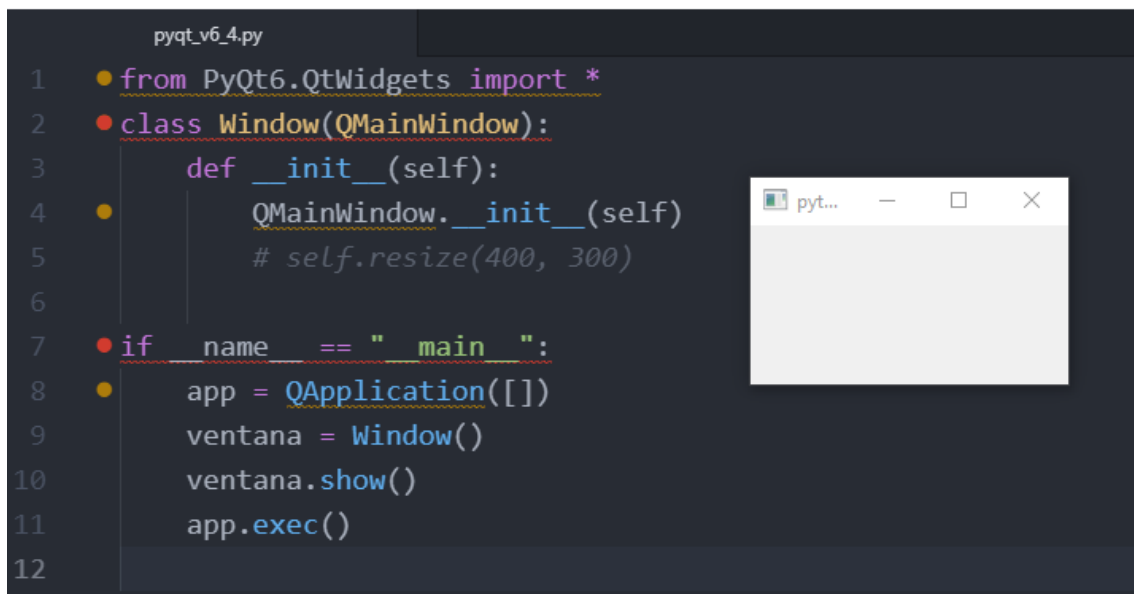


Figura 7.1: PyQt6 con POO

EJEMPLO 1

Podríamos hacer un código muy simple en el que añadiremos algunos parámetros

```

pyqt_v6.5.py x
1  # setStyleSheet() sirve para establecer los estilos
2  • from PyQt6.QtWidgets import *
3  • class Window(QMainWindow):
4      def __init__(self):
5          • QMainWindow.__init__(self)
6          self.resize(500, 400)
7          # label
8          • self.label = QLabel(self)
9          • self.label.setGeometry(30,30,140,50)
10         self.label.setText("Label en POO")
11         • self.label.setStyleSheet("background-color: blue; color: white; margin: 2px; font-size: 20 px;")
12         self.label.setObjectName("label_1")
13         # button
14         • self.button = QPushButton(self)
15         • self.button.setGeometry(30,100,120,50)
16         self.button.setText("button en POO")
17         • self.button.setStyleSheet("font-size: 15px; color: black; background-color: grey")
18         self.button.setObjectName("button_1")

```

Figura 7.2: QSS en PyQt6 -Ejemplo 1 (Parte 1)

Aclaraciones de la anterior figura:

- | Debemos usar self.button o self.label al estar programando bajo el paradigma de la Programación Orientada a Objetos (POO)
- | Hemos seleccionado el ObjectName para cada elemento.

```

19         # line_edit
20         • self.line_edit = QLineEdit(self)
21         • self.line_edit.setGeometry(30,170,140,30)
22         self.line_edit.setStyleSheet("border: 2px solid #3232C0;")
23         self.line_edit.setObjectName("line_edit_1")
24
25     • if name == " main ":
26         app = QApplication([])
27         ventana = Window()
28         ventana.show()
29         app.exec()

```

Figura 7.3: QSS en PyQt6 -Ejemplo 1 (Parte 2)

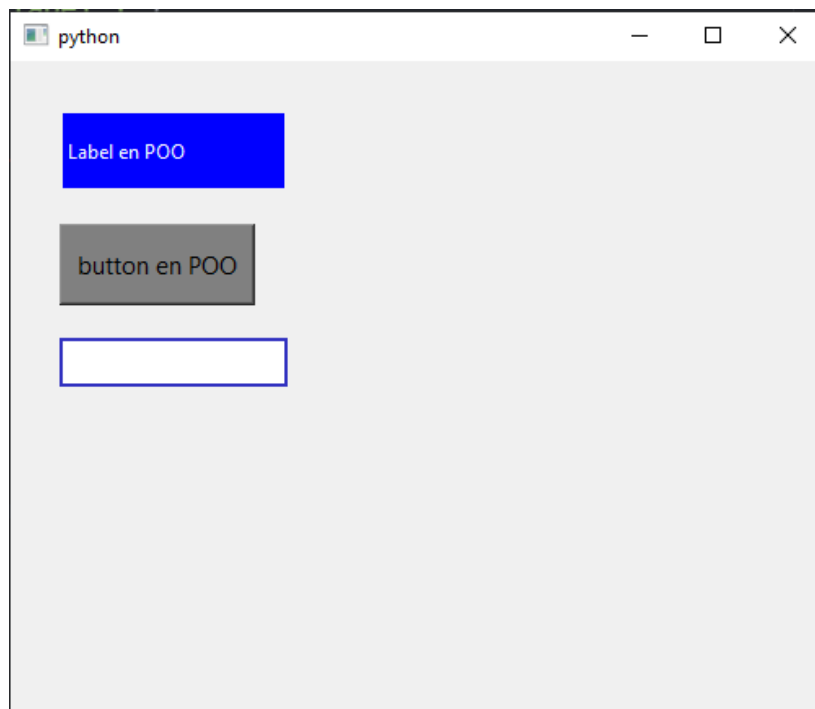


Figura 7.4: QSS en PyQt6 -Ejemplo 1 -Solución

Incluso, podríamos hacer otro ejemplo:

EJEMPLO 2:

```
pyqt_v6_6.py
1  import sys
2  from PyQt6.QtWidgets import *
3
4  if __name__ == '__main__':
5      app = QApplication(sys.argv)
6      ventana = QMainWindow()
7      ventana.setGeometry(800,100,500,500)
8      ventana.setWindowTitle('QSS aplicado a varios elementos')
9      ventana.setStyleSheet("background-color: grey;")
10
11     boton_1 = QPushButton('botón 1', ventana)
12     boton_1.move(100, 100)
13
14     boton_2 = QPushButton('botón 2', ventana)
15     boton_2.move(100, 200)
16
17     boton_3 = QPushButton('botón 3', ventana)
18     boton_3.setStyleSheet("background-color: green; color: white;")
19     boton_3.move(100, 300)
```

Figura 7.5: QSS en PyQt6 -Ejemplo 2 (Parte 1)

Aclaraciones de la anterior figura:

- | Si nos fijamos en los botones 1 y 2 no tenemos seleccionada la QSS, al contrario que en el botón 3

```
20
21     estilos = """QPushButton {
22         background-color: grey;
23         color: black;
24         border-width: 5px;
25         border-style: solid;
26         border-radius: 5;
27         padding: 2px;
28     }"""
29
30     ● QApplication.instance().setStyleSheet(estilos)
31     ventana.show()
32     sys.exit(app.exec())
```

Figura 7.6: QSS en PyQt6 -Ejemplo 2 (Parte 2)

Aclaraciones de la anterior figura:

- | Si nos fijamos: “estilos” es la QSS que aplicaremos a QPushButton (a todos ellos a la vez) a excepción del botón 3 que ya tiene su propio QSS
- | Hemos fijado unas cuantas propiedades para que se aprecie un contraste en el diseño

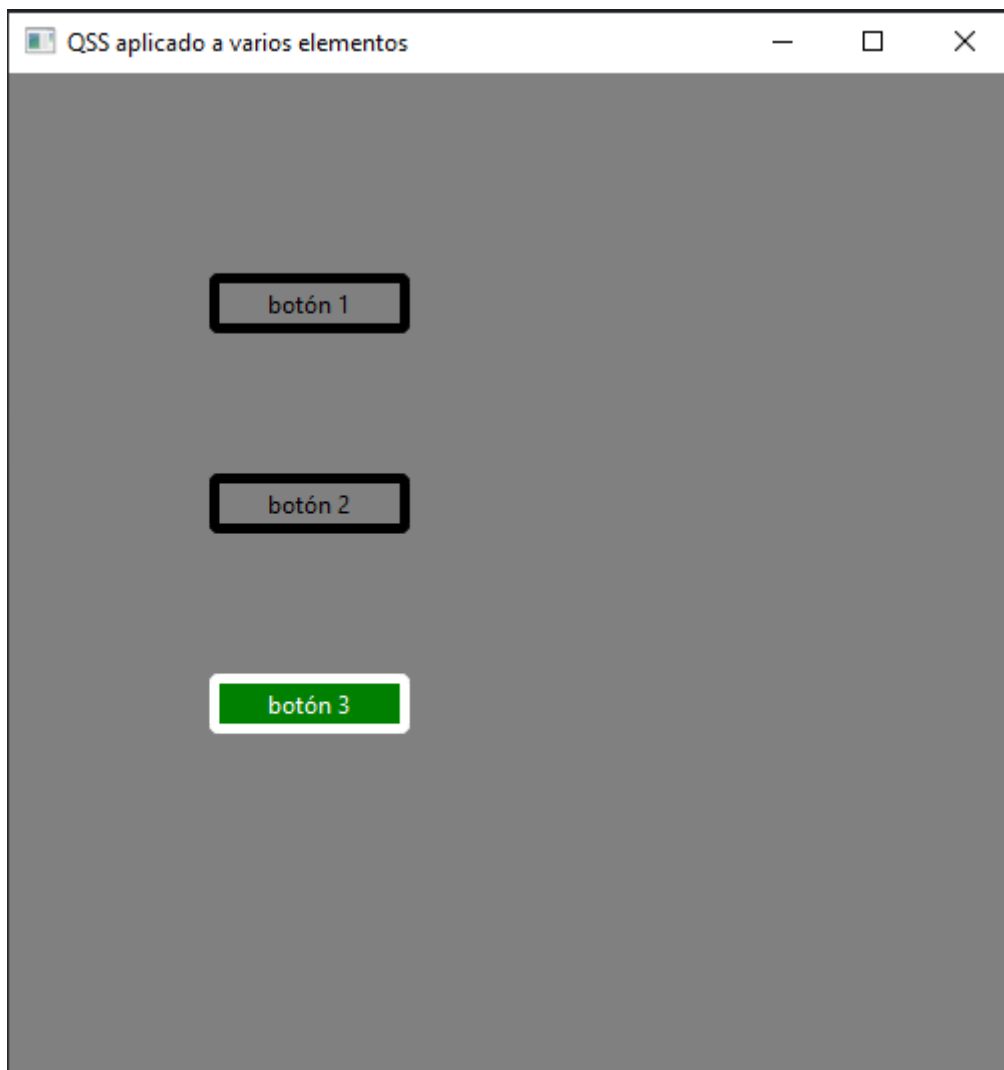


Figura 7.7: QSS en PyQt6 -Ejemplo 2 – Solución

Aclaraciones de la anterior figura:

- | Si nos fijamos los botones 1 y 2 tienen un diseño concreto, y el botón 3 tiene otro distinto, con distintos colores entre otras cosas.

8. PUNTOS CLAVE

- | PyQt es un Framework GUI que permite hacer aplicaciones en Python
- | PyQt6 es la versión más reciente de PyQt (Abril 2021)
- | Qt Designer puede usarse con PyQt y puede ahorrar tiempo.

